National Research University Higher School of Economics Faculty of Computer Science

Programme' Master of Data Science'

MASTER'S THESIS

Detecting patterns in purchase history using association rule learning methods

Student:

David Schmid

Supervisor:

Anastasia Maksimovskaya

Moscow, 2023

ABSTRACT

Table of Contents

[ABSTRACT 2](#_Toc119756171)

[DEFINITIONS, DESIGNATIONS AND ABBREVIATIONS 4](#_Toc119756172)

[1. Introduction 4](#_Toc119756173)

[2 Baseline algorithm 5](#_Toc119756174)

[2.1 Apriori 5](#_Toc119756175)

[2.2 FP-Growth 5](#_Toc119756176)

[3 New Algorithm based on FP-Growth 5](#_Toc119756177)

[4 Conclusion 5](#_Toc119756178)

[5 References 5](#_Toc119756179)

[6 Applications 5](#_Toc119756180)

DEFINITIONS, DESIGNATIONS AND ABBREVIATIONS

**ARL** — Association Rule Learning

**TAR** — Traditional Association Rules

**AR** — Association Rules

**DL** — Machine Learning

**ML** — Machine Learning

1. Introduction

The concept of association rules (AR) was popularized due to a 1993 article by Agrawal et al. [1]. This article shows the application of AR, namely the "apriori-algorithm" for shelf management to maximize profit. In the evolution of basquet data mining, the application of AR belongs to the early methods, which are simple but powerful. AR has not lost its significance in the research. More efficient and modified versions of the base algorithm are developed to mine the relevant and significant basket relations more efficiently. Compared to more complex ML or DL models, the most significant advantage of this relatively simple algorithm is its ease of use and generalization of pre-defined rules, making it scalable to millions of items and Big Data. Moreover, it is simple enough to explain to Business conceptionally.

The main driver for Business is ultimately not the frequency of items but the profitability. One can assume that more frequent items are lower in price and higher in margin but can be equal with not frequent but highly-priced articles. In one article, one reason association rules are not often applied is the lack of relevance. Frequency is only one part, but not the ultimate driver for Business. The interference of profit and frequency could be solved by post-application of associative rules when connecting all frequencies with profit from articles later. However, in tradtional association rules (TAR), we have a **first problem**: we already sorted out the least frequent items and lost crucial relations.

Moreover, so this approach would not be that straightforward. Instead, the author suggests considering profit already within the algorithm as decision criteria, whether to keep the item for getting relational rules.

**The second problem** not considered in TAR is that items can occur multiple times within one transaction. The support counting does not work in such a scenario. Even such a scenario is dependent on the nature of items quite frequently.

The author suggests adding the profit of these items and dividing by the unique count of these items, leading to a new average profit per item. In such a way, we consider these weights in the profit but not in the count, keeping the AR's primal structural integrity based on simple counts of each item per transaction.

**A third problem** poses the question: Why should we consider all transactions with the exact counting if more recent items have higher relevance than older items?

The author suggests introducing a date-decay function to consider the relevance of more recent items.

**The fourth and last problem** the author addresses is algorithm efficiency. We are primarily interested in the itemsets themselves; instead of repetitions leading to the same items. Even the improved FP-Tree algorithm has to loop through the whole dataset many times to find the exact relation between the identified itemset branches.

The author suggests that by simplifying counting directly from the original tree structure (one loop through the whole dataset), we can derive the complete itemsets already with the most count to the minor count of items (1 path). Mostly the other paths are repetitions and do not have many gains. That approach will make the algorithm super fast, with some loss of information (loss of other less relevant paths).

2 Baseline algorithm

## 2.1 Simple dataset

For demonstrating the method of the basic associative algorithm, a simple dataset, the author uses the following dataset.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | I0 | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 | I9 | I |
| T0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 7 |
| T1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 7 |
| T2 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 4 |
| T3 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 7 |
| T4 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 5 |
| T5 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 5 |
| T6 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 3 |
| T7 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 4 |
| T8 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 |
| T9 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 3 |
| T | 8 | 7 | 4 | 4 | 5 | 3 | 2 | 4 | 6 | 5 | 48 |

## 2.2 Formal model of association rules

The author takes the terms and general description of the association rules model by Agrawal et al. [1].

Let I be the   
Let   
Each item is binary. If an item is in the itemset, it is denoted by 1; if missing by 0.  
Let T be the database/dataset of transactions.  
Let   
Let X be a set of items in each transaction.  
A transaction Tn satisfies X, if all items of X commonly occur in Tn.

2.2.1 Support

The number of elements of X is 10.  
1. If .  
T satisfies X in 8 transactions.  
Therefore the support of X is 0.8.  
2. If .  
T satisfies X in 7 transactions.  
Therefore the support of X is 0.7.  
3. If .  
T satisfies X in 5 transactions.

If X consists of at least two items, we have two paths to get to itemset X.  
Agrawal et al. [1] name the existing itemset "antecedent" and the newly added item "consequent."

2.2.2 Paths:

Suppose we have an itemset with support 2. The number of paths leading to an itemset is factorial n!. Therefore for 2 items, there are 2 paths.

First path:, where I0 is the antecedent, and I1 is the consequent, building the itemset X.  
Second path:, where I1 is the antecedent, and I0 is the consequent, building the itemset X.

### 2.2.3 Confidence:

The metric confidence is calculated by dividing the itemset's support by the antecedent's support.

First path's confidence:  
Support of ; Support of

Second path's confidence:  
Support of ; Support of

We see there is not the same confidence for both paths, even if they lead to the same itemset X.

### 2.2.4 Application of association rules to the simple dataset

We can construct and filter out all association rules depending on the itemset's minimum support and confidence.

Suppose we want to find all itemsets with:

Notice that for calculating confidence, X has to consist minimum of 2 items.

**Frequent itemset:**

Text

Description automatically generated with medium confidence

For illustration, the author chose an enough high support to filter only 6 frequent itemsets. If there is no min\_support, there will be 308 itemsets.

**Association rules:**

A screenshot of a computer

Description automatically generated with medium confidence

When choosing confidence as min threshold, only itemsets of at least 2 items get filtered out because single items have no confidence. The confidence threshold has been chosen to be small enough to show all possible combinations of the 3 frequent itemsets with 2 items:

{I\_0,I\_1},  
{I\_0,I\_8},  
{I\_1,I\_9}

For each of the 3 itemsets of exactly 2 items, there exist each 2 paths; in total, 3 \* 2 = 6 possible combinations.

If no minimum support is chosen, 308 itemsets build the basis for the rules. If no confidence threshold is chosen, these 308 itemsets generate 5,183 rules. When considering the very simple dataset T, introduced in chapter 2., it is overwhelming. That reflects problem 1 (relevance) and problem 4 (efficiency and order) described in the introduction.

## 2.3 Apriori

Apriori, Eclat, and FP-Growth, all the TAR lead all to the same result of frequent itemsets as well as to the same rules. There is no difference in the outcome, but there is a difference in the efficiency. In the following, a short example shall illustrate the difference between the least efficient apriori algorithm and the most efficient FP-growth algorithm.

The author recalculate the outcome from the Python library (apriori) step by step.

**Step 1:**  
Let's consider all frequent itemsets >= 0.5 minimum support.  
{I\_0} Support: 0.8  
{I\_1} Support: 0.7  
{I\_4} Support: 0.5  
{I\_8} Support: 0.6  
{I\_9} Support: 0.5  
This we call **List1**  
Notice: For the first round, 1 loop in the database is enough; simply count all occurrences of all items and compare with minimal support

**Step 2:**  
Building all possible combinations out of List1, gives back the following candidate itemsets 1:  
{I\_0,I\_1},{ I\_0,I\_4},{I\_0,I\_8},{I\_0,I\_9}  
{I\_1,I\_4},{I\_1,I\_8},{I\_1,I\_9}

{I\_4,I\_8},{I\_4,I\_9}  
{I\_8,I\_9}

**Step 3:**  
Repeat Step 1 for candidate itemsets 1 >=0.5 minimum support  
{I\_0,I\_1} Support: 0.5  
{I\_0,I\_8} Support: 0.5  
{I\_1,I\_9} Support: 0.5  
This we call **List2**  
Notice: Instead of simply looping once, for every transaction we need to check all possible candidates to be a subset of the transactions. If there list of candidates growths to several 1,000 and the database has over 1 million rows, it can become very inefficient and slow on a normal Desktop computer (without distributed calulations)

**Step 4:**  
From List 2, we can create a new candidate set, by joining all possible combinations with length of itemset 3 together.  
{I\_0,I\_1,I\_8},{I\_0,I\_1,I\_9},{I\_0,I\_8,I\_9},{I\_1,I\_8,I\_9}

**Step 5:**  
By pruning, which means we build all possible subsets with 2 elements (1 element less than the candidate itemset), we can figure out, if the new candidates already disqualify as not above minimum support. If one of the subsets is not in the already found frequent item list, the resulting itemset will not be frequent either. This way, not all possible candidates have to be looped in the database.

Let Frequent = F, Not Frequent = NF  
Subsets for {I\_0,I\_1,I\_8}: {I\_0,I\_1}:F, {I\_0,I\_8}:F, {I\_1,I\_8}: NF -> Disqualified  
Subsets for {I\_0,I\_1,I\_9}: {I\_0,I\_1}:F, {I\_0,I\_9}:NF ,{I\_1,I\_9}: F -> Disqualified  
Subsets for {I\_0,I\_8,I\_9}: {I\_0,I\_8}:F, {I\_0,I\_9}:NF ,{I\_8,I\_9}: NF -> Disqualified  
Subsets for {I\_1,I\_8,I\_9}: {I\_1,I\_8}:NF, {I\_1,I\_9}:F ,{I\_8,I\_9}: NF -> Disqualified

All 4 candidate itemsets contain at least one subset, which is not frequent. Therefore the apriori algorithm will stop here and give back the frequent itemsets and rules. If there are no constraints to any other metrics, the possible rules per itemset are factorial n!.

If there would be at least 1 candidate left, steps 3-5 are repeated until no frequent itemset can be found.

## 2.4 FP-Growth

The FP-Growth algorithm is a big step toward a much more efficient solution of finding frequent itemsets. FP stands for "Frequent Pattern" and it is structured in tree nodes, which grow with each new combination and are dynamically incremented.

The most significant change is that each candidate set has to be looped through the whole dataset. Important to notice is that both algorithms Apriori and FP-Growth don't change the methodology of frequent itemsets and association rules, but have different approaches of getting the result more or less efficient.

**Step 1** is the same as for apriori  
Let's consider all frequent itemsets >= 0.5 minimum support.  
{I\_0} Support: 0.8  
{I\_1} Support: 0.7  
{I\_8} Support: 0.6  
{I\_4} Support: 0.5  
{I\_9} Support: 0.5  
This we call **List1**

Notice: In FP-Growth the outcome of the list is kept in strict order by count. If 2 items have the exact count, it doesn't matter, but most important, once fixed, the order is kept over the whole algorithm.

In Theory, it would be possible to have a random order, which is fixed, or even the least frequent item at the top, and it still would work and lead to the same result. However, we have an upside-down or bizarre-looking tree that would take away some of the efficiency and waste memory and computing power. Therefore there is no reason not to order it from most frequent to least frequent.

Step 2:

The root of the tree is denoted as null.

Building the tree

T0 is filtered by minimum support and ordered by the strictly ordered list of Step 1=

T0 is filtered by minimum support and ordered by the strictly ordered list of Step 1=

The share the same node I0. Therefore, this node is incremented by 1. I4 is a direct child of I0. This combination still does not yet exist and therefore builds a new branch.

After adding all transactions into the tree, the tree looks like the following:

Step 3:  
The frequent itemsets are best gathered by a recursive function and results in the following sets:

Step 4:  
From the itemsets in Step 3, we can build all the frequent itemsets.  
The algorithm with the code implementation is later explained in chapter 4.

Step 5:  
After Step 4 we know all frequent itemsets. By strict logic are all factorial subsets of a frequent itemset themselves frequent too.  
However, all known subsets have to be

3 Discussion of related works

4 Developing a modified Algorithm based on FP-Growth

5 Experiments

6 Conclusion

8 References

1. Agrawal, R.; Imieliński, T.; Swami, A. (1993). "Mining association rules between sets of items in large databases", in: Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93. p. 207

9 Applications

1. Dataset //
2. Algorithm //
3. Analysis //